Formal Methods for Information Security

Luca Dolfi, Selma Steinhoff

May 29, 2019

1 PACE Protocol

The Password-Authenticated Connection Establishment Protocol is used to establish a secure channel between a terminal and an RFID chip on a passport (or ID card). [3]

1.1 A simple challenge-response protocol

The initial protocol P1 is a simple MAC-based challenge response protocol between an initiator A and a responder B where x is a nonce generated by A, $MAC_k(M)$ denotes the MAC of a message M with key k, and k(A, B) is a uni-directional, symmetric long-term key shared by A and B.

$$A \to B : x \tag{1}$$
$$B \to A : MAC_{k(A,B)}(x) \tag{2}$$

We modeled the MAC function by defining an equation in Tamarin that takes the message m and key k as input and returns a pair containing the message itself and a tag which is the message encrypted with the given key. For the encryption we used the built-in cryptographic primitive *symmetric-encryption*. The symmetric shared key is modeled to be distributed and stored in the corresponding agent state once a pair of agents registers themselves.

The first part of the protocol is formalized by a send rule for A and a corresponding receive rule for B. The nonce is freshly generated by A and sent in plain text, which is modeled by the $Out()$ fact. After step (1) both agents, as well as the attacker, have knowledge of the nonce x.

The step (2) of the protocol is modeled by a send rule for agent B in which the mac function is called with the received value x and the previously registered, shared symmetric key. The result of the function call is sent to A. Additionally, we labeled the sent rule of agent B with the $Running(R, I, <' I',' R', x >)$ action. This will help us analyze authentication properties of the protocol. The final receive rule for agent A is labeled with a corresponding $Commit(I, R, <' I',' R', x >)$ action as well as $Honest()$ facts for both agents and a $Finish()$ fact, which helps us analyze if the protocol is executable.

We prove that A injectively agrees with B on the nonce x with the help of the *Running* and *Commit* actions. Specifically, we want the *Commit* action on the parameters $<' I',' R', x >$ to imply that either agent B is running a session with the same parameters or that the adversary has revealed the long-term key of a participant that was assumed to be honest. Additionally, no second session with the same parameters is allowed. We can verify the explained authentication lemma with Tamarin.

The agreement from B point of view is not analyzed, as it was not required for this part.

1.2 Mutual authentication

We interleave two instances of the protocol P1 discussed in Section 1.1 such that the senders alternate. Again, the symmetric keys are uni-directional, so $k(B, A)$ indicates the key A uses to communicate with B, and $k(A, B)$ is the key B uses to communicate with A. A schematic overview of the protocol P2a can be seen below:

$$A \rightarrow B : x \tag{3}$$
$$B \rightarrow A : y \tag{4}$$
$$A \rightarrow B : MAC_{k(B,A)}(y) \tag{5}$$
$$B \rightarrow A : MAC_{k(A,B)}(x) \tag{6}$$

The property we would like to achieve is the agreement of each role with the other role on both nonces x and y. However, inspection of the agreement lemma for the initiator and responder shows that our protocol is vulnerable to a Man-in-the-Middle attack. An attacker could for example in step (3) send its own nonce e to B. B's nonce y reaches A normally as described in step (4) and A will respond with the MAC of y (5). B's response, which contains the MAC of the attackers nonce e, will be blocked and the protocol execution will get stopped at this point. The attacker has successfully tricked B into believing he is running the protocol with A and that he has agreed with A on the nonces e and y, even though A sent the nonce x.

In order to prevent this attack, we modified the protocol to include the names of the communication partner in each message. This did not suffice to prevent an attack where one of the nonces was provided by the attacker. Only after adding both nonces x and y to the MAC, we were able to prove mutual agreement on both nonces for both points of view. A schematic overview of our modified protocol Pb can be seen below:

$$A \rightarrow B : x \tag{7}$$
$$B \rightarrow A : y \tag{8}$$
$$A \rightarrow B : MAC_{k(B,A)}(A, B, y, x) \tag{9}$$
$$B \rightarrow A : MAC_{k(A,B)}(B, A, x, y) \tag{10}$$

With the modified protocol P2b the attack described above is no longer possible. An attacker cannot forward the messages without being detected and he cannot modify the names of the communication partners (without being noticed) as they are maced with the shared symmetric long-term key. In order to prevent attacks where one of the nonces is provided by the attacker, we needed to include both nonces in the MAC for message (9) and (10). This way the protocol participants can detect if they are using nonces that were not provided by the other agent or not intended for this protocol run and abort the protocol. With these modifications agreement on both nonces x and y is reached for both roles with each other.

1.3 Introducing a session key

We replace the long-term key $k(A, B)$ with a session key which is derived from the long-term key and the nonces x and y in the following way: $Kab = kdf(k(A, B), x, y)$. The key derivation function *kdf* is modeled by defining a ternary function in Tamarin. A schematic overview of the protocol P3a can be seen below:

$$A \to B : x \tag{11}$$
$$B \to A : y \tag{12}$$
$$A \to B : MAC_{Kab}(A, B, y) \tag{13}$$
$$B \to A : MAC_{Kab}(B, A, x) \tag{14}$$

We are able to prove that both roles, the initiator and the responder, mutually agree on both nonces x and y and the session key Kab even thought the nonces are not being maced. This is possible because the key which is used to compute the MAC does not only depend on long-term key k(A,B), which is shared between the two agents A and B, but also on the previously exchanged nonces. The attack where one of the nonces is provided by an attacker is therefore no longer possible because two different session keys would be computed and the protocol could not be successfully executed until the end. However, we still require the agent names to be part of the message as explained in Section 1.2. Without these indicators of sender and intended receiver an attacker could reflect the initiators messages back to himself or alternatively send his own nonce e and trick the initiator into believing he is agreeing with B on the two nonces even though B has no knowledge of the current execution.

In order to verify the secrecy of the session key Kab, we introduce the facts *Honest()* and *Secret()*. At the end of an execution, we require the session key to be unknown to the attacker unless one of the agents revealed its long-term key. In that case the attacker can simply compute the session key himself as he knows all the inputs to the key derivation function (the nonces x and y are exchanged in plain-text over a public channel).

In general, we can see that agreement on the nonces is reached in this protocol because otherwise two different sessions keys would be computed by the communication partners which would be noticed during the exchange of the maced nonces. This is a key difference to P2 in which we used the long-term key to mac the nonces that remained unchanged over multiple executions of the protocol and did not depend on the current execution (i.e. on the nonces).

1.4 Replace the password by a nonce

We replace the password $k(A, B)$ in the session key by a nonce s generated by A and add the symmetric encryption of s with the hashed password $h(k(A, B))$ as a second component to the first message from A to B. The session key is now computed in the following way: $Kab = kdf(s, x, y)$. A schematic overview of the protocol P4 can be seen below:

$$A \to B : x, Enc_{h(k(A,B))}(s) \tag{15}$$
$$B \to A : y \tag{16}$$
$$A \to B : MAC_{Kab}(A, B, y) \tag{17}$$
$$B \to A : MAC_{Kab}(B, A, x) \tag{18}$$

We used our model of protocol P3 from Section 1.3 and updated the send and receive rule for the first message to incorporate the change from using a shared password $k(A, B)$ to using a nonce s for the key-derivation function. For the encryption of the nonce s with the hashed password $h(k(A, B))$ we used the built-in cryptographic primitives *symmetric-encryption* and *hashing*.

For our updated protocol model P4, we were able to prove mutual agreement for both roles, the initiator and the responder, on both nonces x and y and the session key Kab and the secrecy

of the session key Kab using Tamarin. This is what we expected as the model of protocol P4 is very similar to the model discussed in the previous section. Both, the password and the nonce s are modeled as fresh values in Tamarin. The main difference to our model of P3 is that in P4 the third input for the key-derivation function (the nonce s) is no longer pre-distributed but gets generated and shared by the initiator in the first step of the protocol. However, as the nonce is encrypted with the password, the attacker does not learn its value. No other modifications to our model were required.

1.5 Introducing Diffie-Hellman: The PACE protocol

We replace the nonces x and y by Diffie-Hellman half-keys g^x and g^y where g is defined by $g = map(s, p)$ and p is a public domain parameter which is added as a plaintext component to the first message. We use the built-in cryptographic primitives *diffie-hellman* for exponentiation and model the mapping function *map* as a binary function in Tamarin. A schematic overview of the protocol P5ab can be seen below:

$$A \to B : g^x, Enc_{h(k(A,B))}(s), p \tag{19}$$
$$B \to A : g^y \tag{20}$$
$$A \to B : MAC_{Kab}(A, B, g^y) \tag{21}$$
$$B \to A : MAC_{Kab}(B, A, g^x) \tag{22}$$

Using the same approach as for protocol P4, we were able to prove that both roles, the initiator and the responder, mutually agree on both half-key g^x and g^y and on the session key $Kab = hash(g^{xy})$. To reach this, we did not need to modify our model other than described in the previous paragraph. Also the property of secrecy of the session key remains true for all traces.

By using Diffie-Hellman instead of nonces, we would like to archive the property of *perfect forward secrecy* for our session key Kab, which means that even if the long-term key gets revealed later on, all encrypted messages with this session key remain secret, as the this key is short-lived and never used again. We find that also this lemma holds without any further modification of our model.

In this protocol, the base g is secret as it is computed by both agents A and B based on the secret, shared nonce s. However, the discrete logarithm is assumed to be computationally hard [2], meaning that even with a publicly known base an attacker could still not determine the exponents x or y given the half-keys and the session key Kab would therefore remain secret.

For protocol P5d we removed any tags from our protocol which made the last two messages unifiable: $MAC_{Kab}(\textit{half-key})$. With this modified protocol, we were able to find an attack on the agreement property for the point of view of the initiator. The initiator can be tricked into believing he is agreeing with his protocol partner on the half-keys and the session key while in reality an attacker simply reflected his messages back to him. However, by requiring the two half-keys to be different this attack can be prevented.

Once we modified our model to require the two half-keys g^x and g^y to differ, the property of mutual agreement from the point of view of the initiator held again. Since the values of x and y are *random* nonces, it is safe to assume that in a normal execution A and B will not produce two identical half-keys. The other properties (agreement for responder, secrecy and perfect forward secrecy of the session key) were not affected by the unifiability of the two last messages.

2 OTR Protocol

The Off-the-Record Messaging (OTR) protocol is designed to add end-to-end security and privacy to Instant Messaging (IM) protocols. It consists of two phases, an authenticated key-exchange to obtain a shared session key and a second phase in which the session key gets continuously refreshed while exchanging messages. [1] In the following we focus only on the first phase of the protocol — the OTR authenticated key-exchange.

2.1 Modeling the original OTR Key Exchange

In preparation for the key exchange both protocol participants generate their Diffie-Hellman half-keys g^x and g^y, which we model using the built-in *diffie-hellman* primitive. These half-keys will be exchanged over a public channel such that both participants can afterwards compute the session key $K = h(g^{xy})$. In order to prevent an attacker from impersonating a protocol participant, the half-keys are signed by the sender and the receiver can check their authenticity based on the *public-key infrastructure*. A schematic overview of the protocol OTR1 can be seen below:

$$A \to B : g^x, Sign_{skA}(\text{"1"}, g^x), pkA \tag{23}$$

$$B \to A : g^y, Sign_{skB}(\text{"2"}, g^y), pkB \tag{24}$$

The protocol is modeled using two rules for each step, one describing the sending and one the receiving of the corresponding message. Even though the paper only specifies each participant to send the signature of its half-key and its public key, we decided to add the half-key as a first, plain-text component to the message and added increasing numbers as strings to help distinguish the messages more easily (meaning the message in step (23) is tagged with the string "1" and the message in step (24) with "2"). Since the content of signatures and therefore also the half-keys are not secret, this does not change the protocol but only helps us modeling it.

By adding increasing numbers as tags, we exclude simple reflection attacks in Tamarin that would easily be detected by the receiver in reality (e.g. since the sent public key matches his own and the signature could have only been created by himself unless his private key was revealed). For example, the tool will no longer show traces in which agent A accepts his own message from step (23) as a reply in step (24) since the strings don't match the expected strings.

After receiving a message, the validity of the signature is always checked by opening it with the public key (3rd component of message) and comparing the content of the signature to the sent half-key (1st component of the message). The message will only be accepted if the two are equal and if the public key belongs to the intended communication partner. These restrictions are reasonable since agents are capable of checking equality and since signature verification is a standard operation in each public-key infrastructure (PKI). Additionally, we require the private keys in the PKI to be unique (meaning there exist no two agents that have the same secret key). Having two agents with the same private key is essentially equal to revealing the private key of an agent since the private key no longer uniquely identifies one agent —i.e. two agents can potentially generate the same signature.

When analyzing our protocol OTR1 with the above explained restrictions in Tamarin, we found the protocol to be executable and the resulting trace was the one presented in the protocol description in Section 2.1 of the OTR paper. Therefore, we argue that it correctly models the described protocol from the paper [1].

2.2 Authentication Failure

Section 3.1 of [1] explains how the authentication can fail in the presence of an attacker. In order to find the exact attack trace with Tamarin that was described in the paper, we used the injective agreement lemma for the initiator as provided in the project description [3]. By adding restrictions to our Tamarin model, which we described in the Section 2.1, we exclude unwanted attacks until the tool finds the attack shown in the schematic overview below:

$$A \to E[B] : g^x, Sign_{skA}(g^x), pkA \tag{25}$$
$$E \to B : g^x, Sign_{skE}(g^x), pkE \tag{26}$$
$$B \to E : g^y, Sign_{skB}(g^y), pkB \tag{27}$$
$$E[B] \to A : g^y, Sign_{skB}(g^y), pkB \tag{28}$$
$$\tag{29}$$

Additionally to the restriction of uniqueness of the private key and correctness of the signature which were explained in Section 2.1, we need to limit the number of initiators in our protocol to one. Without this restriction, we ran into the problem that a larger attack was found where four agents were initiated and the output of the communication between two agents C and D was used as input to agent B whose output was in turn used as an input to agent A. Once we restricted the number of initiators in our protocol to one, we could observe the described attack in which an agent E, whose private key gets revealed (becomes the attackers knowledge), plays Man-in-the-Middle and forwards the reply of his communication with B to agent A. Agent A *commits* on the agent names A and B of roles *initiator* and *responder* and term g^{xy}, meaning A is tricked into believing to have agreed with B on the key g^{xy}. This is exactly the attack presented in Section 3.1 of the paper [1].

2.3 Improvement

In this section we incorporate the improvement described in Section 3.1 [1] and analyze the authentication properties of the resulting protocol OTR3 which is shown below:

$$A \to B : g^x, Sign_{skA}(\text{"1"}, g^x, B), pkA \tag{30}$$
$$B \to A : g^y, Sign_{skB}(\text{"2"}, g^y, A), pkB \tag{31}$$

When analyzing non-injective and injective agreement on the shared session key g^{xy} for the initiators point of view, respectively on the message content g^x, B for the responders point of view, we can observe that the modified protocol now holds for these lemmas. As this is a two-message protocol, by the time agent A receives all information needed to call *Running()* on the session key, B will already have finished the protocol and can no longer *Commit()*. We therefore had to use the message content of the first message for the responder's point of view. If we were to perform the same attack as described in Section 2.2, agent A would receive a message containing the information that the message was intended for agent E and simply discard it. A Man-in-the-Middle attack is no longer feasible since the intended recipient is included in the message and this part of information is signed. The attacker cannot use said message to trick other agents.

2.4 SIGMA

For the last improvement, we modeled the "SIGMA" authenticated key exchange that has been proposed in Section 4 of [1]. Additionally to the signed Diffie-Hellman half-keys (now both keys will get signed), the identities of the sender and a MAC will be added to the last two messages. The MAC key is also derived from the key g^{xy} but computationally independent from the session key. A schematic overview of the protocol OTR4 can be seen below:

$$A \to B : g^x \tag{32}$$
$$B \to A : g^y \tag{33}$$
$$A \to B : \text{'A'}, Sign_{skA}(g^y, g^x), MAC_{Kab}(\text{'0'}, \text{'A'}), pkA \tag{34}$$
$$B \to A : \text{'B'}, Sign_{skB}(g^x, g^y), MAC_{Kab}(\text{'1'}, \text{'B'}), pkB \tag{35}$$

Starting from our model of protocol OTR3, we modified the last two messages to include the additional components. Specifically, we added a string of the identity of the sender and a MAC of counter and said identity to the message. An important difference to the protocol OTR3 is that both half-keys will get signed and returned to the communication partner and that the identity of the intended recipient is no longer part of the signature. The MAC key fully depends on the Diffie-Hellman half-keys and does no longer contain or depend on a shared, long-term key. These modifications have large implications. For example, the identity of the communication partner is not revealed (at least explicitly) in the message.

Instead of strengthening the authentication, we were able to find attacks for non-injective and injective agreement on the key g^{xy} from both points of view. An attacker could for example impersonate the agent B for A and trick A into agreeing on a session key with him. In detail, A would initialize a protocol run with E[B] and send his half-key g^x in step (32). After receiving A's half-key, the attacker starts a normal protocol execution with agent B where he simply uses A's half-key g^x as his own. After a successful protocol run with B, the attacker has knowledge of B's half-key g^y and has a valid signature of B on both half-keys. The attacker can use these information to continue his attack on agent A. He uses the half-key g^y returned by B as his own half-key towards A and use B's last message in step (35) to trick A —in step (35) of that run of the protocol— into believing that A is agreeing with B on the session key and the half-keys; while B has no knowledge of a protocol run with A.

References

[1] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 81–89. ACM, 2005.

[2] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[3] R. Sasse and Ch. Sprenger. Project assignment, Formal Methods for Information Security, 2019.